CUPRINS

8. *PL/SQL – Trigger*-i

- Un *trigger* (declanşator) este un bloc *PL/SQL* cu nume, stocat în baza de date (independent), care se execută automat ori de câte ori are loc evenimentul „declanşator“ de care este asociat.
- Evenimentul declanşator poate consta din:
  - modificarea unei tabele sau a unei vizualizări;
  - acţiuni sistem;
  - acţiuni utilizator.
- Un *trigger* poate fi asociat cu un eveniment care are loc asupra unei tabele, unei vizualizări, unei scheme sau unei baze de date.

❖ Faţă de subprogramele stocate *trigger*-ii:
- pot fi activaţi sau dezactivaţi;
- nu pot fi invocaţi explicit;
- nu pot conţine comenzile *LCD COMMIT*, *SAVEPOINT* sau *ROLLBACK*.

❖ Un *trigger* activ (*enable*) va fi invocat automat de către sistem ori de câte ori are loc evenimentul asociat acestuia.

❖ Un *trigger* dezactivat (*disable*) nu va fi declanşat, chiar dacă evenimentul asociat are loc.

În mod asemănător pachetelor, *trigger*-i nu pot fi definiţi local în blocuri *PL/SQL* sau pachete.

❖ Ca şi în cazul subprogramelor independente sau al pachetelor, atunci când un *trigger* este depus în dicţionarul datelor alături de codul sursă este depusă şi şi forma compilată (*p-codul*).

❖ Dacă *trigger*-ul este valid, atunci va fi apelat fără recompilare.

❖ *Trigger*-ii pot fi invalidaţi în aceeaşi manieră ca pachetele şi subprogramele. Dacă *trigger*-ul este invalidat, acesta va fi recompilat la următoarea activare.

Când utilizăm *trigger*-i?

Atunci când dorim ca efectuarea unei anumite operaţii să implice întotdeauna execuţia unor acţiuni asociate.

- ❖ Nu trebuie definiţi *trigger*-i care duplică sau înlocuiesc acţiuni ce pot fi implementate mai simplu. De exemplu, nu are sens să fie definiţi *trigger*-i care să implementeze regulile de integritate ce pot fi definite prin constrângeri declarative.
- ❖ Utilizarea excesivă a *trigger*-ilor poate determina interdependenţe complexe ce pot fi dificil de menţinut.
- ❖ Atunci când sunt definiţi *trigger*-i trebuie să se ţină cont de recursivitate şi de efectele în cascadă.

- *Trigger*-ii pot fi definiţi la nivel de:
  - aplicaţie (*application triggers*);
  - bază de date (*database triggers*).

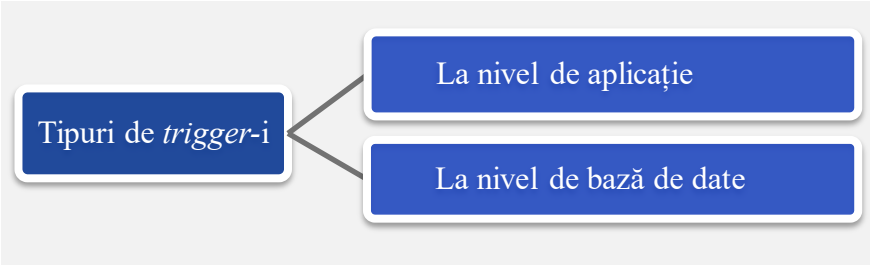

Fig. 8.1. Tipuri de *trigger*-i

Trigger-i aplicaţie

- Se pot executa automat ori de câte ori apare un anumit eveniment într-o aplicaţie (de exemplu, o aplicaţie dezvoltată cu *Developer Suite*).
  - *Form Builder* (utilitar *Developer Suite*) utilizează frecvent acest tip de *trigger*-i (*form builder triggers*). Aceştia pot fi declanşaţi prin apăsarea unui buton, prin navigarea pe un numit câmp etc.

Trigger-i bază de date

- Se pot executa automat ori de câte ori are loc:
  - o comandă *LMD* asupra datelor unei tabele;
  - o comandă *LMD* asupra datelor unei vizualizări;
  - o comandă *LDD* (*CREATE*, *ALTER*, *DROP*) referitoare la anumite obiecte ale schemei sau ale bazei de date;
  - un eveniment sistem (*SHUTDOWN*, *STARTUP*);

- o acţiune a utilizatorului (*LOGON*, *LOGOFF*);
- o eroare (*SERVERERROR*, *SUSPEND*).

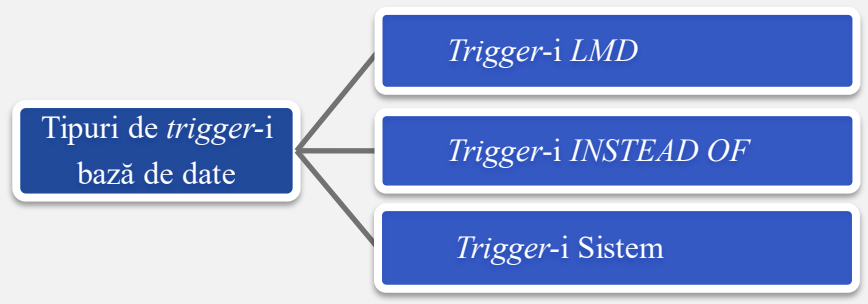

Fig. 8.2. Tipuri de *trigger*-i bază de date

- *Trigger*-ii bază de date sunt de trei tipuri:
  - *trigger*-i *LMD*
    - sunt activaţi de comenzi *LMD* (*INSERT*, *UPDATE* sau *DELETE*) executate asupra unei tabele al bazei de date
  - *trigger*-i *INSTEAD OF*
    - sunt activaţi de comenzi *LMD* executate asupra unei vizualizări (relaţionale sau obiect)
  - *trigger*-i sistem
    - sunt activaţi de un eveniment sistem (oprirea sau pornirea bazei), de comenzi *LDD* (*CREATE*, *ALTER*, *DROP*), de conectarea/deconectarea unui utilizator
    - sunt definiţi la nivel de schemă sau la nivel de bază de date

- ❖ *Trigger*-ii asociaţi unei tabele vor acţiona indiferent de aplicaţia care a efectuat operaţia *LMD*.
- ❖ Dacă operaţia *LMD* se referă la o vizualizare, *trigger*-ul *INSTEAD OF* defineşte acţiunile care vor avea loc, iar dacă aceste acţiuni includ comenzi *LMD* referitoare la tabele, atunci *trigger*-ii asociaţi acestor tabele sunt şi ei, la rândul lor, activaţi.
- ❖ *Trigger*-ii asociaţi unei baze de date se declanşează pentru fiecare eveniment, pentru toţi utilizatorii.
- ❖ *Trigger*-ii asociaţi unei scheme sau unei tabele/unei vizualizări se declanşează numai dacă evenimentul declanşator implică acea schemă sau acea tabelă/acea vizualizare.

8.1. *Trigger*-i *LMD*

- Sunt activaţi de comenzi *LMD* (*INSERT*, *UPDATE* sau *DELETE*) executate asupra unei tabele a bazei de date.
- În comanda de creare a unui *trigger* pot fi precizate mai multe comenzi declanşatoare diferite, dar care se execută asupra unei singure tabele.
- Sintaxa:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
--momentul când este declanşat
  {BEFORE | AFTER}
--comanda/comenzile care îl declanşează
  { DELETE|INSERT|UPDATE [OF coloana[, coloana …] ] }
  [OR {DELETE|INSERT|UPDATE [OF coloana[, coloana …] ] …}
  ON [schema.]nume_tabelă

[REFERENCING {OLD [AS] vechi NEW [AS] nou
              | NEW [AS] nou OLD [AS] vechi } ]
[FOR EACH ROW]
[WHEN (condiţie) ]
  corp_trigger (bloc anonim PL/SQL sau comanda CALL);
```

  - Numele *trigger*-ului:
    - trebuie să fie unic printre numele *trigger*-ilor din cadrul aceleaşi scheme;
    - poate să coincidă cu numele altor obiecte ale schemei în care este definit (de exemplu, tabele, vizualizări sau proceduri).
  - Momentul declanşării *trigger*-ului:
    - reprezintă momentul în care va fi executat corpul *trigger*-ului;
    - poate fi înainte (*BEFORE*) sau după (*AFTER*) comanda declanşatoare;
  - Comanda declanşatoare:
    - poate fi specificată o singură comandă *LMD* (*INSERT*, *DELETE* sau *UPDATE*) sau o combinaţie disjunctivă celor trei comenzi *LMD* (folosind operatorul logic *OR*);
    - poate fi urmată de o enumerare a coloanele a căror actualizare va declanşa *trigger*-ul (comanda declanşatoare este un *UPDATE*).

- Tabela asupra căreia este executată comanda declanşatoare poate fi:
  - o tabelă (*table*);
  - un tablou imbricat (*nested table*).
- Valorile coloanelor înainte şi după modificarea unei linii:
  - Înainte de modificare valoarea unei coloane este referită prin atributul *OLD*, iar după modificare este referită prin atributul *NEW*.
  - În interiorul blocului *PL/SQL*, coloanele prefixate prin *OLD* sau *NEW* sunt considerate variabile externe, deci trebuie prefixate cu ":".
- Clauza *REFERENCING* permite redenumirea atributelor *NEW* şi *OLD*.
- Clauza *FOR EACH ROW* declară un *trigger* la nivel de linie.
  - Lipsa acestei clauze determină definirea unui *trigger* la nivel de instrucţiune.
- Clauza *WHEN* precizează o expresie *booleană* care este verificată pentru fiecare linie corespunzătoare din tabelă.
  - Este valabilă doar pentru *trigger*-ii la nivel de linie.
- Corpul *trigger*-ului
  - nu poate depăşi 32*KB*;
    - pentru a evita depăşirea dimensiunii maxime se pot defini proceduri stocate ce pot fi invocate din corpul *trigger*-ului;
  - poate consta dintr-un bloc *PL/SQL*;
  - poate consta dintr-o singură comandă *CALL*:
    - comanda *CALL* poate apela un subprogram *PL/SQL* stocat;
    - în acest caz, comanda *CALL* nu poate conţine clauza *INTO* care este specifică funcţiilor;
    - pentru a referi coloanele tabelei asociate *trigger*-ului, acestea trebuie prefixate de atributele *:NEW* sau *:OLD*;
    - în expresia parametrilor nu pot să apară variabile *bind*.

- ❖ Un *trigger* poate activa alt *trigger*, iar acesta la rândul său poate activa alt *trigger* etc (*trigger*-i în cascadă).
- ❖ Sistemul permite maximum 32 de *trigger*-i în cascadă.
- ❖ Numărul acestora poate fi limitat (utilizând parametrul de iniţializare *OPEN_CURSORS*), deoarece pentru fiecare execuţie a unui *trigger* trebuie deschis un nou cursor.

- *Trigger*-ii *LMD* pot fi:
  - la nivel de comandă (*statement level trigger*);
  - la nivel de linie (*row level trigger*).

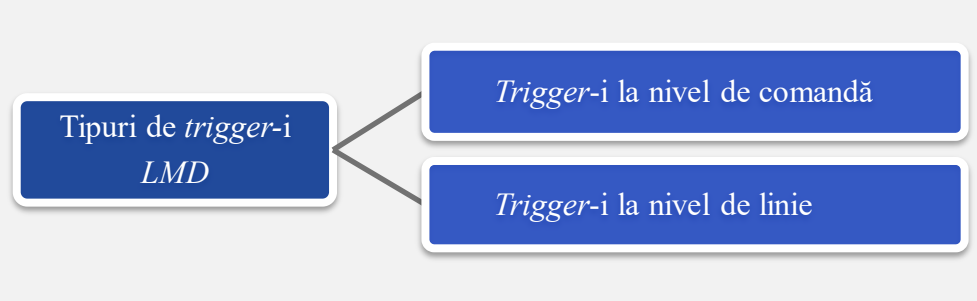

Fig. 8.3. Tipuri de *trigger*-i *LMD*

8.1.1. *Trigger*-i *LMD* la nivel de comandă

- Sunt executaţi o singură dată pentru comanda declanşatoare, indiferent de numărul de linii afectate (chiar dacă nicio linie nu este afectată).
- Sunt utilizaţi atunci când acţiunea *trigger*-ului nu depinde de informaţiile menţinute în liniile afectate de comandă.

Exemplul 8.1 **– vezi curs**

Exemplul 8.2 **– vezi curs**

8.1.2. *Trigger*-i *LMD* la nivel de linie

- Sunt creaţi atunci când în comanda *CREATE TRIGGER* este precizată clauza *FOR EACH ROW*.
  - Lipsa acestei clauze determină definirea unui *trigger* la nivel de instrucţiune.
- Sunt executaţi pentru fiecare linie afectată de instrucţiunea declanşatoare.
  - Dacă instrucţiunea declanşatoare nu afectează nicio linie, atunci nu sunt executaţi.
- Restricţiile acestor tipuri de *trigger*-i pot fi specificate în clauza *WHEN* (printr-o expresie *booleană*).
  - Expresia *booleană* este evaluată de *trigger* pentru fiecare linie afectată de comanda declanşatoare.
  - *Trigger*-ul este executat pentru o anumită linie, doar dacă expresia este adevărată pentru acea linie.
  - În expresia *booleană* nu sunt permise funcţii definite de utilizator sau subcereri *SQL*.

❖ *Trigger*-ii la nivel linie nu sunt performanţi dacă se fac frecvent reactualizări pe tabele de dimensiuni foarte mari.

Exemplul 8.3 **– vezi curs**

- Accesul la vechile, respectiv noile valori ale coloanelor liniei curente, afectată de evenimentul ce a declanşat *trigger*-ul, se realizează prin:
  - OLD.nume_coloană
  - NEW.nume_coloană

❖ În interiorul blocului *PL/SQL*, coloanele tabelei prefixate cu *OLD* sau *NEW* sunt considerate variabile externe şi deci, trebuie precedate de caracterul „:“.

❖ În expresia *booleană* din clauza *WHEN* nu trebuie utilizată prefixarea cu „:“ pentru *OLD* sau *NEW*.

- În cazul celor trei comenzi *LMD*, aceste valori devin:

COMANDA	NEW.nume_coloană	OLD.nume_coloană
INSERT	noua valoare	null
DELETE	null	vechea valoare
UPDATE	noua valoare	vechea valoare

Exemplul 8.4 **– vezi curs**

8.1.3. Ordinea de execuţie a *trigger*-ilor *LMD*

- *PL/SQL* permite definirea a 12 tipuri de *trigger*-i *LMD* care sunt obţinuţi prin combinarea proprietăţilor ce pot fi specificate în comanda de definire a acestora:
  - momentul declanşării (*BEFORE* sau *AFTER*);
  - nivelul la care acţionează (nivel comandă sau nivel linie – *FOR EACH ROW*);
  - comanda declanşatoare (*INSERT*, *UPDATE* sau *DELETE*).

❖ De exemplu, un *trigger BEFORE INSERT* acţionează o singură dată, înaintea execuţiei unei instrucţiuni *INSERT*, iar un *trigger BEFORE INSERT FOR EACH ROW* acţionează înainte de inserarea fiecărei noi înregistrări.

- O comandă declanşatoare sau o comandă din corpul unui *trigger* poate determina verificarea mai multor constrângeri de integritate. De asemenea, *trigger*-ii pot conţine comenzi care pot determina declanşarea altor *trigger*-i (*trigger*-i în cascadă).
- Pentru a menţine o secvenţă adecvată de declanşare a *trigger*-ilor şi de verificare a constrângerilor de integritate, sistemul *Oracle* utilizează un model de execuţie a *trigger*-lor *LMD* multipli. De exemplu, în cazul unei comenzi *update* modelul este:
  1. Se execută toţi *trigger*-ii *BEFORE* comandă care sunt declanşaţi de comanda *LMD*.
  2. Pentru fiecare linie afectată de comanda *LMD*:
    2.1. se execută toţi *trigger*-ii *BEFORE* linie care sunt declanşaţi de comanda *LMD*;
    2.2. se blochează şi se modifică linia afectată de comanda *LMD* (se rulează comanda *LMD*); se verifică constrângerile de integritate (blocarea rămâne valabilă până în momentul în care tranzacţia este permanentizată);
    2.3. se execută toţi *trigger*-ii *AFTER* linie care sunt declanşaţi de comanda *LMD*.
  3. Se execută toţi *trigger*-ii *AFTER* comandă care sunt declanşaţi de comanda *LMD*.

- Definiţia modelului de execuţie este recursivă.
  - De exemplu, o comandă *SQL* poate determina execuţia unui *trigger BEFORE* linie şi verificarea unei constrângeri de integritate. Acel *trigger BEFORE* linie poate realiza o actualizare (*UPDATE*) care la rândul său determină verificarea unei constrângeri de integritate şi execuţia unui *trigger AFTER* comandă. *Trigger*-ul *AFTER* comandă determină la rândul său o verificare a unei constrângeri de integritate.
  - În acest caz, modelul de execuţie rulează paşii recursiv, după cum urmează:
    Este lansată comanda *SQL* declanşatoare.
    1. Se execută *trigger*-ii *BEFORE* linie.
      1.1. Se execută *trigger*-ii *AFTER* comandă declanşaţi de comanda *UPDATE* din corpul *trigger*-ului *BEFORE* linie.
        1.1.1. Se execută comenzile din corpul *trigger*-ilor *AFTER* comandă.
        1.1.2. Se verifică dacă sunt îndeplinite constrângerile de integritate definite pentru tabelele modificate de *trigger*-ii *AFTER* comandă.
      1.2. Se execută comenzile din corpul *trigger*-ilor *BEFORE* linie.
      1.3. Se verifică dacă sunt îndeplinite constrângerile de integritate definite pentru tabelele modificate de *trigger*-ii *BEFORE* linie.

2. Se execută comanda *SQL*.
3. Se verifică dacă sunt îndeplinite constrângerile de integritate ce ar putea fi încălcate de comanda *SQL*.

- Există două excepţii privind recursivitatea.
  - Atunci când o comandă modifică o tabelă (cheia primară sau cheia externă) care face parte dintr-o constrângere referenţială şi declanşează un *trigger* ce modifică cealaltă tabelă referită în constrângere, doar comanda declanşatoare va determina verificarea constrângerii de integritate. Astfel, se permite *trigger*-ilor la nivel de linie să forţeze integritatea referenţială.
  - *Trigger*-ii la nivel de comandă declanşaţi datorită opţiunilor *DELETE CASCADE* şi *DELETE SET NULL* sunt executaţi înainte şi după lansarea comenzii *DELETE* de către utilizator. În acest mod se previne apariţia erorilor *mutating*.

- O proprietate importantă a modelului de execuţie a *trigger*-ilor este acela că toate acţiunile şi verificările realizate datorită execuţiei unei comenzi declanşatoare trebuie să se termine cu succes.
- Dacă apare o excepţie în interiorul unui *trigger* şi aceasta nu este explicit tratată, atunci toate acţiunile realizate ca rezultat al comenzii declanşatoare, incluzând toate acţiunile realizate de *trigger*-ii declanşaţi de comandă vor fi anulate (*rollback*).
- În acest mod, *trigger*-i nu pot compromite constrângerile de integritate.

- *Trigger*-ii de tipuri diferite sunt executaţi într-o ordine specifică.
- *Trigger*-ii de acelaşi tip definiţi pentru aceeaşi comandă nu au garantată o ordine specifică. De exemplu, toţi *trigger*-ii *BEFORE* linie definiţi pentru o singură comandă *LMD* nu sunt declanşaţi mereu în aceeaşi ordine. Din acest motiv, în aplicaţii nu se recomandă utilizarea mai multor *trigger*-i de acelaşi tip care sunt declanşaţi de aceeaşi comandă.

- Se poate specifica ordinea de execuție a *trigger*-ilor de acelaşi tip definiţi pentru aceeaşi comandă?

- *Trigger*-ii bază de date pot fi definiţi numai pe tabele (excepţie, *trigger*-ul *INSTEAD OF* care este definit pe o vizualizare).

- ❖ Totuşi, dacă o comandă *LMD* este aplicată unei vizualizări, pot fi activaţi *trigger*-ii asociaţi tabelelor care definesc vizualizarea.

Restricţii:

- ❖ În versiunile *Oracle* anterioare corpul unui *trigger* nu poate conţine o interogare sau o reactualizare a unei tabele aflate în plin proces de modificare, pe timpul acţiunii *trigger*-ului (*mutating table*). Începând cu versiunea *Oracle 11g* această problemă este rezolvată prin utilizarea *trigger*-ilor compuşi (*compound triggers*).
- ❖ Blocul *PL/SQL* care descrie acţiunea *trigger*-ului nu poate conţine comenzi pentru gestiunea tranzacţiilor (*COMMIT*, *ROLLBACK*, *SAVEPOINT*).
  - Controlul tranzacţiilor este permis, însă, în procedurile stocate.
  - Dacă un *trigger* apelează o procedură stocată care execută o comandă referitoare la controlul tranzacţiilor, atunci va apărea o eroare la execuţie şi tranzacţia va fi anulată.
- ❖ Comenzile *LDD* nu pot să apară decât în *trigger*-ii sistem.
- ❖ În corpul *trigger*-ului pot fi referite şi utilizate coloane *LOB*, dar nu pot fi modificate valorile acestora.
- ❖ În corpul *trigger*-ului se pot insera date în coloanele de tip *LONG* şi *LONGRAW*, dar nu pot fi declarate variabile de acest tip.
- ❖ Dacă o tabelă este eliminată, automat sunt eliminați toţi *trigger*-ii asociaţi acesteia.

8.1.4. Predicate condiţionale

- În interiorul unui *trigger* care poate fi executat pentru diferite tipuri de instrucţiuni *LMD* se pot folosi trei funcţii *booleene* (din pachetul *DBMS_STANDARD*) prin care se stabileşte tipul operaţiei executate:
  - *INSERTING* – întoarce valoarea *TRUE* atunci când comanda declanşatoare este o comandă *INSERT*;
  - *DELETING* – întoarce valoarea *TRUE* atunci când comanda declanşatoare este o comandă *DELETE*;
  - *UPDATING* – întoarce valoarea *TRUE* atunci când comanda declanşatoare este o comandă *UPDATE*;
    - *UPDATING*('nume_coloană') întoarce *TRUE* atunci când comanda

declanşatoare este o comandă *UPDATE* asupra coloanei *nume_coloană*.

❖ Utilizând aceste predicate, în corpul *trigger*-ului se pot executa secvenţe de instrucţiuni diferite, în funcţie de tipul operaţiei *LMD* declanşatoare.

Exemplul 8.5 **– vezi curs**

8.2. *Trigger*-i *INSTEAD OF*

- Oferă o modalitate de actualizare a vizualizărilor obiect şi a celor relaţionale.
- Sintaxa:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
--momentul când este declanşat
 INSTEAD OF
--comanda/comenzile care îl declanşează
 { DELETE|INSERT|UPDATE [OF coloana[, coloana …] ] }
 [OR {DELETE|INSERT|UPDATE [OF coloana[, coloana …] ] …}
 ON
    [schema.]nume_vizualizare
  [REFERENCING {OLD [AS] vechi NEW [AS] nou
              | NEW [AS] nou OLD [AS] vechi } ]
FOR EACH ROW
[WHEN (condiţie) ]
 corp_trigger (bloc anonim PL/SQL sau comanda CALL);
```

- *Trigger*-ul *INSTEAD OF* permite reactualizarea unei vizualizări prin comenzi *LMD*.
- Datorită regulilor stricte existente pentru reactualizarea vizualizărilor, nu orice tip de vizualizare permite reactualizări *LMD*.
  - De exemplu, o vizualizare care este definită pe baza *join*-ului mai multor tabele nu permite reactualizarea tuturor acestor tabelelor.
  - O vizualizare nu poate fi modificată prin comenzi *LMD* dacă aceasta conţine operatori pe mulţimi, funcţii grup, clauzele *GROUP BY*, *CONNECT BY*, *START WITH* sau operatorul *DISTINCT*.
- *Trigger*-ul *INSTEAD OF* este utilizat pentru a executa operaţii *LMD* direct pe tabelele de bază ale vizualizării.
  - Comenzile *LMD* lansate asupra unei vizualizări, sunt preluate de *trigger*-ul *INSTEAD OF* (care poate lansa comenzile direct pe tabelele de bază).

- *Trigger*-ul *INSTEAD OF* poate fi definit asupra vizualizărilor ce au drept câmpuri tablouri imbricate, *trigger*-ul furnizând o modalitate de reactualizare a elementelor tabloului imbricat.
  - În acest caz, *trigger*-ul se declanşează doar în cazul în care comenzile *LMD* operează asupra tabloului imbricat (numai când elementele tabloului imbricat sunt modificate folosind clauzele *THE*() sau *TABLE*()) şi nu atunci când comanda *LMD* operează doar asupra vizualizării.
  - Atributele :*OLD* şi :*NEW* se referă la liniile tabloului imbricat, iar pentru a referi linia curentă din tabloul „părinte“ s-a introdus atributul :*PARENT*.

- Spre deosebire de *trigger*-ii *LMD*, *trigger*-ii *INSTEAD OF* se execută în locul instrucţiunii *LMD* (*INSERT*, *UPDATE*, *DELETE*) specificate.
- Opţiunea *UPDATE OF* nu este permisă pentru acest tip de *trigger*.
- *Trigger*-ii *INSTEAD OF* se definesc pentru o vizualizare, nu pentru o tabelă.
- *Trigger*-ii *INSTEAD OF* acţionează la nivel de linie.

Exemplul 8.6 **– vezi curs**

8.3. *Trigger*-i sistem

- Pot fi definiţi la nivelul:
  - bazei de date;
  - schemei.
- Sunt declanşaţi de:
  - comenzi *LDD* (*CREATE*, *DROP*, *ALTER*);
  - evenimente sistem (*STARTUP*, *SHUTDOWN*, *LOGON*, *LOGOFF*, *SUSPEND*, *SERVERERROR*).
- Sintaxa:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
  {BEFORE | AFTER}
  {comenzi_LDD | evenimente_sistem}
  ON {DATABASE | SCHEMA}
  [WHEN (condiţie) ]
  corp_trigger;
```

- Pentru *trigger*-ii sistem se pot utiliza funcţii speciale care permit obţinerea de informaţii referitoare la evenimentul declanşator.

- Sunt funcţii *PL/SQL* stocate care trebuie prefixate de numele proprietarului (*SYS*):
  - *SYSEVENT* – întoarce evenimentul sistem care a declanşat *trigger*-ul (este de tip *VARCHAR2*(20) şi este aplicabilă oricărui eveniment);
  - *DATABASE_NAME* – întoarce numele bazei de date curente (este de tip *VARCHAR2*(50) şi este aplicabilă oricărui eveniment);
  - *SERVER_ERROR* – întoarce codul erorii a cărei poziţie în stiva erorilor este dată de argumentul de tip *NUMBER* al funcţiei (este de tip *NUMBER* şi este aplicabilă evenimentului *SERVERERROR*);
  - *LOGIN_USER* – întoarce identificatorul utilizatorului care a declanşat *trigger*-ul (este de tip *VARCHAR2*(30) şi este aplicabilă oricărui eveniment);
  - *DICTIONARY_OBJ_NAME* – întoarce numele obiectului referit de comanda *LDD* care a declanşat *trigger*-ul (este de tip *VARCHAR2*(30) şi este aplicabilă evenimentelor *CREATE*, *ALTER*, *DROP*).

8.3.1. *Trigger*-i pentru comenzi *LDD*

- Sunt declanşaţi de comenzi *LDD* (*CREATE*, *ALTER*, *DROP*).
  - *ON DATABASE* determină declanşarea *trigger*-ului de comenzi *LDD* aplicate asupra obiectelor din orice schemă a bazei de date;
  - *ON SCHEMA* determină declanşarea *trigger*-ului de comenzi *LDD* aplicate asupra obiectelor din schema personală.
- Există restricţii asupra expresiilor din condiţia clauzei *WHEN*.
  - De exemplu, *trigger*-ii *LDD* pot verifica tipul şi numele obiectelor definite, identificatorul şi numele utilizatorului.

Exemplul 8.7 **– vezi curs**

8.3.2. *Trigger*-i pentru evenimente sistem

- Sunt declanşaţi de anumite evenimente sistem (*STARTUP*, *SHUTDOWN*, *LOGON*, *LOGOFF*, *SUSPEND*, *SERVERERROR*).
  - *ON DATABASE* determină declanşarea *trigger*-ului de evenimente sistem pentru orice schemă a bazei de date;
  - *ON SCHEMA* determină declanşarea *trigger*-ului de evenimente sistem din schema personală.

- Există restricţii asupra expresiilor din condiţia clauzei *WHEN*.
  - De exemplu, *trigger*-ii *LOGON* şi *LOGOFF* pot verifica doar identificatorul (*userid*) şi numele utilizatorului (*username*).

Exemplul 8.8 **– vezi curs**

Exemplul 8.9 **– vezi curs**

Exemplul 8.10 **– vezi curs**

8.4. Modificarea şi ştergerea *trigger*-ilor

Ştergerea *trigger*-ilor

- Similar procedurilor şi pachetelor, un *trigger* poate fi:
  - şters folosind comanda

```
DROP TRIGGER [schema.]nume_trigger;
```

  - recreat folosind clauza *OR REPLACE* din cadrul comenzii *CREATE TRIGGER*
    - clauza permite schimbarea definiţiei unui *trigger* existent fără suprimarea acestuia.

Modificarea *trigger*-ilor

- Modificarea unui *trigger* poate consta din:
  - recompilare (*COMPILE*);
  - redenumire (*RENAME*);
  - dezactivare (*DISABLE*);
  - activare (*ENABLE*).
- Sintaxa:

```
ALTER TRIGGER [schema.]nume_trigger
  {ENABLE | DISABLE | COMPILE | RENAME TO nume_nou};
```

- Uneori în loc de ştergerea unui *trigger* este preferabilă doar dezactivarea sa temporară.
  - Un *trigger* este activat implicit atunci când acesta este creat.
  - Un *trigger* dezactivat continuă să existe ca obiect în dicţionarul datelor, dar este nu va mai fi executat de sistem (până când nu este din nou activat).
- Activarea, respectiv dezactivarea tuturor *trigger*-ilor asociaţi unei tabele se poate realiza utilizând comanda următoare:

```
ALTER TABLE [schema.]nume_tabelă
  {ENABLE | DISABLE } ALL TRIGGERS;
```

- Comanda *ALTER TRIGGER* permite activarea, respectiv dezactivarea unui singur *trigger* (al cărui nume este specificat în comandă).
- Comanda *ALTER TABLE* permite activarea, respectiv dezactivarea tuturor *trigger*-ilor asociaţi unei tabele.

- Comanda *DROP TRIGGER* permite ştergerea unui singur *trigger* (al cărui nume este specificat în comandă).
- Comanda *DROP TABLE* determină implicit ştergerea tuturor *trigger*-ilor asociaţi unei tabele.

8.5. Informaţii despre *trigger*-i

- Vizualizări din dicţionarul datelor ce conţin informaţii despre *trigger*-i:
  - *USER_OBJECTS*
    - atunci când este creat un *trigger* în vizualizare apare o linie nouă cu informaţii despre acesta (nume, tip, id, data creării, data ultimei modificări, stare etc);
    - tipul obiectului creat este *trigger*.
  - *USER_TRIGGERS*
    - conţine informaţii complete despre *trigger* (codul sursă detaliat, starea);
  - *USER_TRIGGER_COLS*
    - conţine informaţii despre coloanele utilizate în *trigger*;
  - *USER_ERRORS*
    - conţine informaţii despre erorile apărute la compilare *trigger*-ului;
  - *USER_DEPENDENCIES*
    - este utilizată pentru a determina dependenţele *trigger*-ilor.

Vizualizarea *USER_TRIGGERS*

- Vizualizarea include următoarele informaţii:
  - numele *trigger*-ului (*TRIGGER_NAME*);
  - tipul *trigger*-ului (*TRIGGER_TYPE*);
  - evenimentul declanşator (*TRIGGERING_EVENT*);

- numele proprietarului tabelei (*TABLE_OWNER*);
- numele tabelei pe care este definit *trigger*-ul (*TABLE_NAME*);
  - dacă obiectul referit de *trigger* nu este o tabelă sau o vizualizare, atunci *TABLE_NAME* este are valoarea *null*;
- clauza *WHEN* (*WHEN_CLAUSE*);
- corpul *trigger*-ului (*TRIGGER_BODY*);
- antetul (*DESCRIPTION*);
- starea *trigger*-ului (*STATUS*)
  - poate să fie *ENABLED* sau *DISABLED*;
  - numele utilizate pentru a referi parametrii *OLD* şi *NEW* (*REFERENCING_NAMES*).

- În operaţiile de gestiune a bazei de date este necesară uneori reconstruirea instrucţiunilor *CREATE TRIGGER*, atunci când codul sursă original nu mai este disponibil.
  - Aceasta se poate realiza utilizând informaţiile din vizualizarea *USER_TRIGGERS*.

Exemplul 8.11 **– vezi curs**

8.6. Privilegii sistem

- Sistemul furnizează privilegii sistem pentru gestiunea *trigger*-ilor:
  - *CREATE TRIGGER* (permite crearea *trigger*-ilor în schema personală);
  - *CREATE ANY TRIGGER* (permite crearea *trigger*-ilor în orice schemă cu excepţia celei corespunzătoare utilizatorului *SYS*);
  - *ALTER ANY TRIGGER* (permite activarea, dezactivarea sau compilarea *trigger*-ilor din orice schemă cu excepţia utilizatorului *SYS*);
  - *DROP ANY TRIGGER* (permite suprimarea *trigger*-ilor la nivel de bază de date din orice schemă cu excepţia celei corespunzătoare utilizatorului *SYS*);
  - *ADMINISTER DATABASE TRIGGER* (permite crearea sau modificarea unui *trigger* sistem referitor la baza de date);
  - *EXECUTE* (permite referirea, în corpul *trigger*-ului, a procedurilor, funcţiilor sau pachetelor din alte scheme).

8.7. Tabele *mutating*

- O tabelă *mutating* este o tabelă care este modificată curent de o comandă *LMD* (tabela este aflată în proces de modificare).
- Un *trigger* la nivel de linie nu poate obţine informaţii (*SELECT*) dintr-o tabelă *mutating*, deoarece ar "vedea" date inconsistente (datele din tabelă ar fi în proces de modificare în timp ce *trigger*-ul ar încerca să le consulte).

❖ Tabelele nu sunt considerate *mutating* pentru *trigger*-ii la nivel de comandă.
❖ Vizualizările nu sunt considerate *mutating* în *trigger*-ii *INSTEAD OF*.

❖ Atunci când este definit un *trigger* trebuie respectată următoarea regulă:
Comenzile *SQL* din corpul *trigger*-ului nu pot consulta sau modifica date dintr-o tabelă *mutating*.
❖ Chiar tabela pe care este definit *trigger*-ul este o tabelă *mutating*.
❖ Există o sigură excepţie:
Dacă o comandă *INSERT* afectează numai o înregistrare, *trigger*-ii la nivel de linie pentru înregistrarea respectivă nu tratează tabela ca fiind *mutating*.
❖ Comanda *INSERT INTO nume_tabelă SELECT ...* consideră tabela *mutating* chiar dacă cererea întoarce o singură înregistrare.

❖ Fiecare versiune nouă a bazei de date *Oracle* reduce impactul erorilor *mutating*.
❖ Dacă un *trigger* determină o astfel de eroare, atunci singura opţiune este ca acesta să fie rescris (o soluţie este utilizarea *trigger*-ilor la nivel de comandă).

Exemplul 8.12 **– vezi curs**
Exemplul 8.13 **– vezi curs**
Exemplul 8.14 **– vezi curs**
Exemplul 8.15 **– vezi curs**
Exemplul 8.16 **– vezi curs**
Exemplul 8.17 **– vezi curs**

Implementați cu ajutorul unui *trigger* următoarea restricție: un client poate beneficia într-un an de cel mult 3 perioade cu prețuri preferențiale.

Bibliografie

1. Connolly T.M., Begg C.E., Database Systems: *A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2025
4. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2025
5. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2025
6. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2025
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2023
8. Oracle and/or its affiliates, *Pro*C/C++ Programmer's Guide*, 1996, 2019
9. Oracle University, *Oracle Database*: *PL/SQL Fundamentals*, *Student Guide*, 2009, 2025
10. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004